TITLE

CENTRALIZED TRANSACTIONAL SECURITY AUDIT FOR ENTERPRISE SYSTEMS

REFERENCE TO A SEQUENCE LISTING, A TABLE, OR A COMPUTER PROGRAM LISTING APPENDIX

[0001] A computer program listing is attached as Appendix I.

BACKGROUND OF THE INVENTION

[0002] This invention relates to the art of centralized auditing of authentication, authorization and access control procedures for client computers to access networked servers and services. More particularly, this invention relates to centralized auditing in the technologies of firewalls, intrusion detection systems (IDSs), intrusion prevention systems (IPSs), proxy servers, and multi-server, multi-domain, single-sign-on systems.

[0003] As the impact of the Internet continues to alter the economic landscape, companies are experiencing a fundamental shift in how they do business. Business processes involve complex interactions between companies and their customers, suppliers, partners and employees. Businesses also interact with vendors and suppliers in placing orders and obtaining payments. Businesses must also make a wide array of information and services available to their employees, generating further interactions. One of the key business requirements is to be able to audit the activities of a particular customer to whom the multiple services are provided by service providers. Those service providers can be a single organization or multiple organizations, and those services can be interconnected through both Internet and intranet. The service providers and the services then define a system held together by a computer network.

[0004] Protection of a system may have many components. One of these, authentication, is the process of verifying the identity of a party requesting information from or sending information to the system. Authentication is generally accomplished through the use of passwords.

[0005] Authorization determines whether a user has privileges (viewing, modifying, etc.) with regard to a resource of the system. For instance, a system administrator can determine which users have access to a system and what privileges each user has within the system (i.e., access to certain files, a particular service, etc.). Authorization is usually performed after authentication. In other words, if a user requests access to a system resource, the system will first verify or authenticate the identity of the user and then determine whether that user has the right to access the system's resource. Access control is the mechanism to grant or deny the requestor access to the resources requested based on the results of authentication and authorization.

[0006] Audit logs are records of the sequential activities of a user that constitute trails of that user's activities. These logs also provide protection by enforcing accountability, i.e., by allowing an auditor to trace a user's activities, which are either related to a resource or actually performed on a resource. Audit trails themselves must be secure from unauthorized alterations; for instance, unauthorized users cannot be allowed to remove evidence of their activities from an audit log. Auditing requests and actions generates a huge amount of information; therefore, any system generating audit trails must have the capability to store the information and process it efficiently.

[0007] As service systems become increasingly complex, a service is often required to access multiple sub-services across multiple layers and networks. Thus a centralized audit is required to log all the activities of a particular user during a service transaction and archive the log at a central location for processing that information in a reliable and efficient manner.

[0008] The prior art teaches systems that log access system events. See for example, US publication 20020116642, filed by Joshi et al, and International application WO 0205487 filed by Oblix, Inc. When an access system event occurs, a log entry is created for that access system event. Information from an identity profile is stored in the log entry. The identity profile pertains to a first user. The first user is the entity that caused or was involved with the access system event. In one embodiment, the access system includes identity management and access management functionality.

[0009] However, the above-described process is not transactional and request-response based. The disadvantage of using this method is that it relies too heavily on LDAP (lightweight directory access protocol) or databases for storing the log entry at the logging point. When a transaction requires many layers of services, which can be in multiple tiers or even require access to another server cluster across the Internet, the accessibility of LDAP or databases can be too limiting. On the other hand, frequently accessing databases for writing consumes significant resources and generally degrades the performance of the service significantly. Also, this method cannot be used from an extra-net to access the intra-net, since the access of internal database resources is usually not allowed at by firewall.. Another drawback of this method is that it is fairly complicated to implement.

[0010] There is other method and system in the prior art for authenticating and auditing access by a user to non-natively secured applications. This system is described in US patent 5748890 issued to Goldberg et al. However, this method is not transactional and request-response based and the underlying authentication model is tied to native authentication system. This method cannot provide centralized auditing for multi-tier server systems and across the Internet.

[0011] The prior art also teaches the establishment of log files to create an audit trail, but this method is not useful in single sign-on and multi-tiered service systems. See US patent 5,864,665, issued to Tran. The method also does not include a transaction-based, central auditing capability.

[0012] There is also a single-sign-on process, disclosed in US publication 2002/0099671 filed by Mastin Crosbie et al, for a client who wishes to access multiple servers in an environment where the servers employ the Kerberos authentication process. However, there is no audit trail mechanism for this process.

[0013] The prior art further includes Internet message headers that include a message identifier and a message handling trace header for message handling logging purposes. The standard format for message headers is explained in "Standard for the Format of ARPA Internet Text Messages," by David H Crocker, Department of Electrical Engineering, University of Delaware, which is available at www.freesoft.org/CIE/RFC/822/index.htm. However, these message headers are not suited for security audit trail or use over firewalls, IDS, or proxy servers for security audit of transaction-based enterprise systems. Moreover, it is not clear how these work with an audit-request and audit-response structure for holding the audit information.

[0014] What is needed is to have a method for keeping track of the audit trail right from the entry point of the network – even at the beginning of the firewall – all the way to the point where the response is returned. The method should be implementable using hardware or software. Also, it should be transaction-based since multiple operations and interaction with servers, services and networks can be associated with the request. The method should have the capability of keeping track of all the events, user activities, including exceptions, errors and the audit logs from firewall, IDS, proxy server, web server, application server and workflow. Furthermore, this method should include the current existing authentication, authorization methods to provide a centralized security audit that is specific to a particular user, and should be simple and have minimal impact on the performance of the overall system. This method should be applicable to as many types of existing systems as possible, including firewall, intrusion detection systems, proxy server, web server, application server, messaging services, workflow and mainframe servers over wired or wireless networks.

[0015] The present invention herein described will present a simple and straightforward solution to meet the above requirements.

SUMMARY OF THE INVENTION

According to its major aspects and briefly recited, the present invention is a computer network security system that provides authentication, authorization, access control and centralized audit for a network. In general, the present system verifies the user's identity, grants access rights to the identified user, allows (or denies) the access request by the access controller and provides a centralized audit for all transactions in order to protect the security of the resources available to a network. The system is a request-response based transactional auditing model that provides centralized security auditing over a processing device or sequence of processing devices using the concepts defined in the following steps. At the transaction entry point, a unique Transaction ID, an audit-request object and an audit-response object are created. The Transaction ID is added into the audit-request object, which is next embedded with the incoming request in compliance with the communication protocol that the incoming request is using downstream during the request process. The audit-response object should also carry the Transaction ID. The audit-response object is also embedded with the outgoing response in compliance with the protocol upstream during the response process, whereby the transaction id, audit-request object and audit-response object can be used to identify the transaction and log the events during the request process and response process at any point of the transaction to provide centralized security audit. There is thus no way to differentiate the logged events to which it would belong to without this Transaction ID.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] The following detailed description when taken in conjunction with the figures presented herein provides a complete disclosure of the invention.

[0017] FIG 1 shows the overall architecture of the computer network security system, according to a preferred embodiment of the present invention.

[0018] FIG 2 shows the Computer Block Diagram of the computer network security system, according to a preferred embodiment of the present invention.

[0019] FIG 3 shows the Firewall Block Diagram of the computer network security system, according to a preferred embodiment of the present invention.

[0020] FIG 4 shows the Transactional Audit System of the computer network security system, according to a preferred embodiment of the present invention.

[0021] FIG 5 shows the Security Proxy Server High Level View of the computer network security system, according to a preferred embodiment of the present invention.

[0022] FIG 6 shows the Authentication Server High Level View of the computer network security system, according to a preferred embodiment of the present invention.

[0023] FIG 7 shows the Authentication System of the computer network security system, according to a preferred embodiment of the present invention.

[0024] FIG 8 shows the authentication detailed flowchart for Security Proxy Server of the computer network security system, according to a preferred embodiment of the present invention.

[0025] FIG 9 shows the authentication detailed flowchart for Authentication Server of the computer network security system, according to a preferred embodiment of the present invention.

[0026] FIG 10 shows the User Registry Schema of the computer network security system, according to a preferred embodiment of the present invention.

[0027] FIG 11 shows the Authorization System of the computer network security system, according to a preferred embodiment of the present invention.

[0028] FIG 12 shows the Web Service protection scenario by using customizable plug-in module of the computer network security system, according to a preferred embodiment of the present invention.

DETAILED ESCRIPTION OF PREFERRED EMBODIMENTS

[0029] **FIG 1** depicts a security system that provides authentication, authorization, access control and centralized auditing for a network. In general, the present system verifies the user's identity, grants the identity's access rights, allows or denies the access request by the access controller and provides a centralized audit for all transactions to provide security for the resources available within a network or networks. The transaction-based centralized audit portion of the present system (hereinafter "Transactional Audit System") manages the centralized logs for each transaction. The Transactional Audit System is embedded in a firewall **401 (FIG 4)**, IDS **402 (FIG 4)**, security proxy server **403 (FIG 4)**, web server **404 (FIG 4)**, application server **405 (FIG 4)**, Security Proxy Server **413**, Web Services **414**, MQ server **406 (FIG 4)**, or mainframe server **407 (FIG 4)**. The authentication portion of the present system (hereinafter "the Authentication System") manages an end user authentication process. The Authentication System is incorporated into the Security Proxy Server **105**, the Authentication Server **107** and the User Registry **109**, while the authorization portion of the system (hereinafter "the Authorization System") provides security for resources across one or more web servers, application servers, web services, workflows (not shown), message queue (MQ) servers and mainframe server. The Authorization System is incorporated into Security Proxy Server **105** and embedded in the web services, application servers, web services and mainframe. A key feature of one embodiment of this system is to provide centralized auditing via a request response based transactional audit mechanism disclosed by this invention (**FIG 4**). Another key feature of the system is that the Transaction Audit System has been incorporated with a Kerberos Authentication and Authorization model to provide a multi-server, multi-domain single sign-on for the enterprise. The Kerberos protocol is well known to those skilled in the art of Internet security; version 5 is the default for network authentication for Windows 2000 by Microsoft Corporation, for example. The Transactional Audit System is further employed to provide security for web services, workflow, MQ Server and Mainframe Services. Other features include the special handling of the Web Service Security and a plug-in module mechanism at the Security Proxy Server **105** to allow business customization of the Security Proxy Server **105.**

[0030] **FIG 1** is a block diagram depicting one embodiment for deploying the present system. **FIG. 1** shows web browsers **101** and Client Applications **102** accessing Security Proxy Server **105** via Internet or Intranet **103**. In one embodiment, web browser **101** is standard web browser known in the art and adapted to run on any suitable type of computer. The transmission protocol shown in the figure can be either HTTP or HTTPS. Note that HTTP can only be used in the situation wherein only limited security is required since there is no confidentiality guaranteed during the network transmission. To reach the Security Proxy Server **105**, a request needs to pass through the first tier firewall **104** and an intrusion detection system (IDS) **115**. For a request to reach the Web Server **108,** it has to pass the second firewall **106.** The Authentication Server **107** is also located behind the second firewall **106** to secure the network against unauthorized access to crucial information, like user profile and access policies. Web Server **108** is a standard web server, well known in the art, and provides an end user with access to various resources via Internet **103**. Web Server **108** contains web resources.

[0031] **FIG 2** illustrates a high-level block diagram of a computer system that can be used for the components of the present invention. This computer system can be implemented as a hardware device to host the firewall, IDS, proxy server, web server, and application server mentioned in this invention. Although the firewall and IDS can also be built on dedicated hardware, the present invention will be demonstrated using a general-purpose computer. The computer system of **FIG 2** includes a processor unit **202** and main memory **201**. Processor unit **202** may contain a single microprocessor, or may contain a plurality of microprocessors for configuring the computer system as a multi-processor system. Main memory **201** stores, in part, instructions and data for execution by processor unit **202**. If the system of the present invention is wholly or partially implemented in software, main memory **201** can store the executable code when in operation. Main memory **201** may include banks of dynamic random access memory (DRAM) as well as high-speed cache memory.

[0032] The system of **FIG 2** further includes a mass storage device **203**, peripheral device(s) **204**, user input device(s) **205**, portable storage medium drive(s) **207**, a graphics subsystem **208** and an output display **209**. For purposes of simplicity, the components shown in **FIG 2** are depicted as being connected via a single bus **210**. However, the components may be connected through one or more data transport means. For example, processor unit **202** and main memory **201** maybe connected via a local microprocessor bus, and the mass storage device **203**, peripheral device(s) **204**, portable storage medium drive(s) **207**, and graphics subsystem **208** may be connected via one or more input/output (I/O) buses. Mass storage device **203,** which may be implemented with a magnetic disk drive or an optical disk drive, is a non-volatile storage device for storing data and instructions for use by processor unit **202**. In one embodiment, mass storage device **203** stores the system software for implementing the present invention for purposes of loading to main memory **201**.

[0033] Portable storage medium drive **207** operates in conjunction with a portable non-volatile storage medium, such as a floppy disk, to input and output data and code to and from the computer system of **FIG 2**. In a preferred embodiment, the system software for implementing the present invention is stored on such a portable medium, and is input to the computer system via the portable storage medium drive **207**. One or more peripheral devices **204** may include any type of computer support device, such as an input/output (I/O) interface, to add functionality to the computer system. For example, peripheral device(s) **204** may include a network interface for connecting the computer system to a network, a modem, a router, etc.

[0034] User input device(s) **206** provides a portion of a user interface. User input device(s) **206** may include an alpha-numeric keypad for inputting alpha-numeric and other information, or a pointing device, such as a mouse, a trackball, stylus, or cursor direction keys. In order to display textual and graphical information, the computer system of **FIG 2** includes graphics subsystem **208** and output display **209**. Output display **209** may include a cathode ray tube (CRT) display, liquid crystal display (LCD) or other suitable display device. Graphics subsystem **208** receives textual and graphical information, and processes the information for output to display **209**. Additionally, the system of **FIG 2** includes output devices **209**. Examples of suitable output devices include speakers, printers, network interfaces, and monitors.

[0035] The components contained in the computer system of **FIG 2** are those typically found in computer systems suitable for use with the present invention, and are intended to represent a broad category of such computer components that are well known in the art. Thus, the computer system of **FIG 2** can be configured as a firewall, an intrusion detection system, a proxy server, a personal computer, a workstation, a server, a minicomputer, or any other computing device. The computer can also include different bus configurations, networked platforms, multi-processor platforms, etc. Various operating systems can be used including Unix, Linux, Windows, Macintosh OS, Palm OS, and other suitable operating systems.

[0036] **FIG 3** depicts typical firewall. A *firewall* is a computer, router, or some other communications device that controls data flow between networks. Generally, a firewall is a first-line defense against attacks from the outside world. A firewall can be hardware-based or software-based. A hardware-based firewall is a special router with additional filter and management capabilities. A software-based firewall runs on top of the operating system and turns a computer into a firewall. **Fig 3** depicts a typical firewall with components, such as network policy **301**, advanced authentication mechanisms **302**, packet filtering **303** and application gateways **304**.

[0037] For network policy **301**, there are two levels of network policy that directly influence the design, installation and use of a firewall system. The higher-level policy is an issue-specific, network access policy that defines those services that will be allowed or explicitly denied access to the restricted network, how these services will be used, and the conditions for exceptions to this policy. The lower-level policy describes how the firewall will actually go about restricting the access and filtering the services that were defined in the higher-level policy. The following sections describe these policies in brief.

[0038] Advanced authentication **302** measures such as smart cards, authentication tokens, biometrics, and software-based mechanisms are designed to counter the weaknesses of traditional passwords. While the authentication techniques vary, they are similar in that an attacker who has monitored a connection cannot reuse the passwords generated by advanced authentication devices. Given the inherent problems with passwords on the Internet, an Internet-accessible firewall that does not use or does not contain the *hooks* to use advanced authentication makes little sense.

[0039] IP packet filtering **303** is done usually using a *packet filtering router* designed for filtering packets of data as they pass between the router's interfaces. A packet filtering router usually can filter IP data packets based on some or all of the following fields:

source IP address,
destination IP address,
TCP/UDP source port, and
TCP/UDP destination port.

[0040] To counter some of the weaknesses associated with packet filtering routers, firewalls need to use software applications to forward data while providing filter connections for services such as TELNET and FTP. Such an application is referred to as a *proxy service*, while the host running the proxy service is referred to as an *application gateway*. Application gateways **304** and packet filtering routers can be combined to provide higher levels of security and flexibility than if either were used alone.

[0041] An "intrusion" is an incident caused by someone (such as a "hacker" or "cracker") attempting to break into or misuse a system. The word "misuse" is broad, and can reflect something as severe as stealing confidential data or something minor such as misusing your email system for spam. An "Intrusion Detection System (IDS)" is a system for detecting such intrusions.

[0042] An Intrusion Detection Systems (IDS) monitors packets on the network wire and attempts to discover if a hacker/cracker is attempting to break into a system (or cause a denial-of-service attack). A typical example of IDS is a system that watches for large number of TCP connection requests (SYN) to many different ports on a target machine, thus discovering if someone is attempting a TCP port scan. A Network IDS may run either on the target machine that watches its own traffic (usually integrated with the stack and services themselves), or on an independent machine vicariously watching all network traffic (hub, router, probe). Note that a "network" IDS may monitor many machines, whereas the others monitor only a single machine (the one they are installed on).

[0043] Just like firewall, IDS is a computer, router, or some other communications device that controls data flow between networks. IDS can be hardware-based or software-based. In the present invention, a software based IDS is used to demonstrate a Transactional Audit System. However, this should not limit the scope of this invention.

[0044] As shown in **FIG 1**, Application Server **116** can be a standard J2EE Application Server, well known in the art, and provides the service specified in J2EE. It can also be other types of application servers and not specified by J2EE. The communication between the Web Server **108** and Application Server **116** can be secure socket layer (SSL) enabled or disabled. The Application Server 116 can host multiple applications and multiple web services. A web service is a special type of web application under J2EE. A request can request services provided from other networks **114** including the Internet **114.**

[0045] MQ Server **112** is a standard Message Queue Server known in the art and provides the message queue services. The communication between the Application Server **108** and MQ Server **112** can also be SSL enabled or disabled. The messages can be encrypted and decrypted based on the SSL protocol using a software application.

[0046] The Mainframe Server **113** is a standard Mainframe Server known in the art. Mainframe server can be accessed from the MQ Server **112**

and also from Application Server **116** via HTTP or HTTPS if an Application Server, like **WEBSPHERE** manufactured by IBM, is installed on Mainframe Server **113.**

[0047] The Authentication Server **107** is a gateway for gaining access to User Registry **109** to authenticate a user or an identity. Also, Authentication Server **107** controls the access polices that need to be loaded to Security Proxy Server **105** or any other resources that need to be authorized for a particular policy defined for that resource. For example, those resources can include Web Server **106**, Application Server **116**, Web Services **116** and Mainframe Server **113.** Authentication Server 107 will also provide the Kerberos ticket-granting token (TGT) that contains a user profile and access token to provide single sign-on access across the enterprise.

[0048] The User Registry **109** includes an LDAP Server **110** or Remote Database Server (RMDB) **111.** User Registry **109** contains the user profile and the available roles. Each user may be associated with one or more roles for resource access.

[0049] The system of **FIG 1** is scalable in that there can be many Security Proxy Servers **105**, many Authentication Servers **107**, and many Web Servers **108**. In one preferred embodiment, Directory Server **110** is an LDAP Directory Server and communicates with other servers/modules using LDAP over SSL or other secured protocol. In other embodiments, RMDB **111** can implement other protocols or can be other types of data repositories.

[0050] **FIG 4** depicts the Transactional Audit System. This embodiment is the fundamental part of this invention. The goal of the Transactional Audit System is to solve the problem facing the network security industry today, which is the inability to centralize auditing of all the activities of an identity associated with an enterprise level of transaction. The activities the Transactional Audit System can start the audit trail at the entry point when the identity's request enters the network, for example, at the firewall. It will be able to audit all the events happening during the transaction, either from downstream or upstream. With this Transactional Audit System, organizations will be able to keep track of all the access activities that happen in a multi-tiered enterprise environment and log them at a single location. This system will minimize the overall cost of performing a similar function by current security systems and provide better performance.

[0051] Network technology is made possible by network communication protocols. There are elaborate protocols for communication over the Internet. A protocol is simply a detailed specification of how communication is to proceed. For two processing devices to communicate, they must both be using the same protocol. The most basic protocols on the Internet are the Internet Protocol (IP), which specifies how data is to be physically transmitted from one computer to another, and the Transmission Control Protocol (TCP), which ensures that data sent using IP is received in its entirety and without error. These two protocols, which are referred to collectively as TCP/IP, provide the foundation for communication. Other protocols use TCP/IP to send specific types of information such as files and electronic mail.

[0052] Before getting into the detailed description of the Transactional Audit System, a few concepts require definitions.

[0053] A transaction is a group of sequential operations that will be invoked by one top-level operation. For example, an HTTP request for account transfer may invoke a group of sequential operations, such as database access operations, message queue operations, mainframe operations or other request over the Internet across other network or networks which contains its own firewall, IDS, IDP, proxy server, web server, MQ, application server... or mainframe. A transaction will also include traveling through a workflow within a processing device.

[0054] A transaction entry point is the point to start a transaction. In this invention, since once the connection between the source and destination is established, the firewall or any other security devices simply pass bytes between the systems, thus the transaction entry point can be designated to be anywhere after the incoming data stream is received. The following examples show where the transaction entry point can be designated. But, as those of ordinary skill will know from the following examples, the definition of transaction entry point is broader than the examples shown here:

[0055] TCP/IP or UDP/IP connection point: The transaction entry point is at the connection point when connection is established and the data stream is received.

[0056] A firewall is a barrier that permits incoming authorized messages to pass but not unauthorized messages. For example, there is one type of firewall that will decrypt the secure socket layer (SSL) request message and filter it based on its internal algorithm, then encrypt it and send it to the destination server. The transaction entry point location is at the point at which the firewall decides to pass the request forward.

[0057] An Intrusion Detection System (IDS) detects attempts to obtain access to the resources of a network by those who have not been authenticated. The transaction point can start at the point after the IDS decides a request will be allowed to go forward.

[0058] Security Proxy Servers are used when the network designer wishes for the transaction point to start before the authentication or when the user has been authorized to access a network resource.

[0059] The above examples can be repeated in wireless connections.

[0060] An Object can be anything that may be represented in digital form. An object can have "properties" to represent its characteristics and "methods" to represent its behavior, as is well known in object-based programming. The properties of an object are represented by data structure.

[0061] For example: an object can be just a transaction id such as:

www.bytescreen.com:1051741018650:123456789,

or, in addition to the transaction id, which is a property of the object, it can have an additional property, "logdata", and a method, such as addlog(), to add information into the property logdata.

[0062] An audit-request object is an object for representing audit information during a request process. It should at least contain a transaction id in its audit-data before the audit data is persisted in persistence storage.

[0063] An audit-response object is an object for representing audit information during response process. It should at least contain a transaction id in its audit-data before the audit data is persisted in persistence storage. Note that an Audit-request object can be the same object as Audit-response object but does not necessarily have to be the same.

[0064] A unique id is an id that uniquely identifies an identity. For example, a location parameter and a time parameter plus a random number can produce a unique id. For example:

[0065] Transaction ID is a unique id that will be associated with the transaction.

[0066] Persistent storage device or persistence storage is a digital storage device that temporarily or permanently holds digital data, for example, memory, in-memory database, in-memory cache, hard disk, file, or database as well known in the art.

[0067] Processing device is a device that is capable of processing digital data, for example, a computer, a firewall, an intrusion detection system, a proxy server, a web server, an application server, and a mainframe server.

[0068] HTTP Header is a header defined by HTTP protocol or user defined but satisfying HTTP protocol format, such as the header "Cookie", well known in the art.

[0069] Communication protocols enable communicating over the Internet. A protocol is simply a detailed specification of how communication is to proceed. For two computers to communicate, they must both be using the same protocols. The most basic protocols on the Internet are the Internet Protocol (IP), which specifies how data is to be physically transmitted from one computer to another, and the Transmission Control Protocol (TCP). Others include but not limited to UDP/IP, IIOP, HTTP, SOAP, JMS and TIBCO. The invention here is applicable to any communication protocol especially for above-mentioned protocols.

[0070] XML is Extensible Markup Language, well known in the art.

[0071] SOAP stands for Simple Object Access Protocol, well known in the art.

[0072] The login-count refers to how many login attempts the user has failed.

[0073] Access policy defines the rule of how access to a resource is granted or denied based on the available roles and the roles associated with the user. For example, an access policy could be the following: a given universal resource locator (URL) can be only accessed by the role of Adjuster.

[0074] Workflow:

[0075] Workflow represents business process composed of multiple steps. A workflow can be within the same component/server or across multiple components or servers. For example, a workflow can be in an application server which can contain the utilization of (but not limited to) servlet, EJB, COM object and flow through all those components with business specific steps.

[0076] DOMAIN

[0077] A domain typically represents organization, sub division or a user registry. For example: ibm.com, bytescreen.com, are internet domain names. bytescreen.com can also representing the LDAP organization root for user registry.

[0078] TGT: Kerberos ticket-granting ticket as well known in the art.

[0079] Each enterprise can define its own audit-request object and audit-response object to store the audit information based on a specific type of transaction. In appendix **2001**, a sample XML schema that specifies an XML based audit-request and audit-response object has been listed. Please note that in this case the audit-request object and audit-response object is the same object.

[0080] A sample XML based on this schema is shown in Appendix **2002**.

[0081] The audit data stored in the audit-request object such as in this XML can be passed downstream along with the request, and can also be passed upstream along with the response. But exactly how is protocol-dependent.

[0082] In an HTTP(S) request, the transaction id and audit data can be passed via HTTP header. We can define an HTTP header named "audit-data" or any other non-used HTTP header name. Note that one of the fields in the audit-request object defined in this sample XML is the transaction id, and the XML shown above can be added into the request body as the value of the HTTP header audit-data by following the HTTP protocol specified in the art. The value of the header is composed of a sequence of bytes.

[0083] When the downstream processing system receives the request, the audit-request object stored in the HTTP header "audit-data" can be extracted and new audit information can be added into this XML and then again put into the request body via HTTP header as discussed above.

[0084] The same header can be passed upstream when the response process begins. To ensure security, if the transmission protocol is not a secure protocol such as SSL, the audit data XML should be encrypted before putting it into the audit-data header.

[0085] If the communication protocol was altered during the transaction process, for example, when the Application Server **405 (FIG 4)** sends the request to MQ Server **406 (FIG 4),** the communication protocol can be in Java Message Service Format. In this case the audit-request object and audit-response object can be set in the customized JMS Header fields as specified in the JMS specifications, known in the prior art, to pass it to the receiving end. It is also possible to include the audit-request object together with the message itself, but this should be avoided if possible in order to decouple the security logic and the business logic.

[0086] **FIG 4** depicts a sample deployment to demonstrate a request response based transactional audit flow. The system deployment configuration may vary according to business needs. For a given communication protocol, say HTTP(S) in this example, a request is sent to the enterprise system using the given protocol.

[0087] The block **409** in **FIG 4** depicts common processing steps for a processing node. When a request comes in, it will first be checked to determine if the request is authorized access to its resource. If authorized, the information that needs to be audited is audited and the results of the audit are added to the audit-request object, and the audit-request object will be incorporated into the request body. If persistence of the audited information is required, the audit-request object will be made to persist, then the request to the next processing system will be passed using secured transmission. When the response is received, the audit information will be added into the audit-response object and then the audit-response object will be incorporated into the response body. Then if persistence is required, the audit-response object will be placed into the persistence storage device. Finally, the response to the parent process system with secured transmission will be returned.

[0088] The authorization is processing-system dependent. When the request is in the firewall, the authorization is based on the firewall's authorization logic. When the authorization is in the IDS, the authorization is dependent on IDS's authorization logic. Normally, no user authentication is done within the firewall or IDS, thus the authorization in the firewall or IDS will normally not depend on the user profile or any roles assigned to the user. However, at the Security Proxy Server **105** and downstream, the user will be authenticated and thus the user profile, user assigned roles and user access policies generally will be used to authorize the user's request.

[0089] As shown in **FIG 4**, the request first reaches the outmost firewall **401**. As in this configuration, the transaction entry point is selected at the firewall. The firewall inspects the data, and then decides if the access should be granted or denied. If access is granted, a transaction id is created. The transaction id is added to the audit-request object. Next, the audit-request object is added to the request body via an HTTP header. The request is then sent to IDS **402** via a secured transmission protocol, for example SSL.

[0090] The IDS **402** will process the request as shown in **409** and check if the request should be granted or denied based on its detection logic. If the access is granted, it will add the audit information to the audit-request object, update the HTTP header audit-data value with the modified audit-request object and pass the request to security proxy server **403** via a secured transmission protocol, for example HTTPS.

[0091] The security proxy server will process the request as shown in **409** (in the following sections, a more detailed description about the authentication and authorization process will be given). If authorized, will add audit data into audit-request object and then update the HTTP header audit-data and then pass the request to Web Server **404** via a secured transmission protocol, for example HTTPS.

[0092] The Web Server **404** will process the request as shown in **409** in the standard web server plug-ins (as described in the art), if authorized, then pass the request to Application Server **405** via a secured transmission protocol, such as, for example, SSL.

[0093] In this particular scenario, the Application Server **405** will process the request as shown in **409** and then pass the request to MQ Server **406** and an external network **408** sequentially. Before it passes through the request, it will first authorize the access. If granted, it will add the audit information into the audit-request object and, if the next processing system is MQ Server **406**, add the audit-request object into the JMS message header field or message body and send the request via JMS Message through a secured transmission protocol (such as SSL) to MQ Server **406.**

[0094] If the next processing system in located in the external network, and assuming the communication protocol is HTTPS, the audit-request object will be added into HTTP header audit-data or other applicable header, then incorporated into the request body and passed to the destination server via external network, such as the Internet **408.**

[0095] If the next processing system is a web service **414,** known in the art, the communication protocol is HTTPS. Then the audit-request object is into HTTP header audit-data or other applicable header, then incorporated into request body and passed to the security proxy server **413**. Then the security proxy server will authorize the request and pass the request to the Application Server that is hosting the web service **414.**

[0096] Since MQ Server **406** is a Message Queue Server, it will put the incoming request into the message queue and wait for the MQ client to retrieve the request message.

[0097] On the other end, the external network **408** will start a new transaction process with that network and return the response to the Application Server **405** with the audit information logged with its domain. Note that this external network must use the transaction id created at the transaction entry point of the primary network to log the access information located in its domain. There will be only one transaction id for each transaction.

[0098] The Mainframe Server **407** will process the request as shown in **409**, authorize the request for access and, if granted, allow the request to access the system and log the audit data into the audit-response object. In this case, the audit-response object is the audit-request object (the XML string) obtained from Message Header field.

[0099] The Security Proxy Server **413** will intercept the web service request and then authorize the request for the access. The authorization process will be described in the later part. The Security Proxy Server will retrieve the audit-request object from the HTTP header and log the audit data into the audit-request object. The audit data include authorization status and any desirable data that need to be logged. If the access is granted, the Security

Proxy Server **413** will pass the request to the destination web service hosted by an application server **414.**

[00100] In the Web Service case, there are two options: one is using the HTTP Header to pass the audit-request and audit-response object, and the other is using SOAP over HTTP. With SOAP option, special handling is needed to pass the audit-request object and audit-response object into request and response. First, the audit-request object and audit response object has to be defined by XML. The structure of the XML needs to be defined as an XML schema as known in the art. The schema that defines the audit-request object and audit-response object need to be included within the Web Service request message and response message respectively. During the request process, the Security Proxy Server **413** will insert the audit-request object into the SOAP message for request. When web service **414** implementation receives the request, it will be able to access the audit-request object and add or remove audit data to or from it. And during the response process, web service **414** implementation will extract the data in the audit-request object and use them to populate the audit-response object that is embedded in the response. A software application can then add or remove the audit-data as needed and pass them back to the parent process Security Proxy Server **413**. Security Proxy Server **413** will extract the audit-response object from the SOAP message body and incorporate it into the HTTP header in the response. The SOAP message element that carries the audit-response object needs to be reset as a nil element.

[00101] The response process is also shown in **409.** That is, the audit-response object is obtained, the audit info is added into the audit-response object and then the audit-response object is embedded into the response body so that the response can be returned to the parent processing system.

[00102] A centralized audit point can be selected any given point of the process. For example, the centralized audit point in this deployment is selected at the security Proxy Server **412**. At the centralized audit point, the audit data in the audit-response object will be persisted onto a desired persistence storage device or devices. For example, the audit-response object can be just logged into a file at the security proxy server in an encrypted format. Then the audit-

response object should be removed from the response body. This is important because the audit-response object should not be returned to the client.

[00103] What has been described so far has demonstrated how the Transactional Audit System can be implemented by passing audit-request object and audit-response object in a request-response based transactional system.

[00104] The Transaction Audit System described above can also be used as a data passing or parameter passing mechanism for processing devices to communicate with each other in a structured manner.

[00105] The data that has been persisted into persistence storage devices can also be used as input for subsequent transaction invocation since at those persistence points, data that available is collected data from part of the transaction..

[00106] The data in the persistence storage is in fact a result of data-binding of user profile, accessing policies, and business related information. This data binding mechanism is very important in a enterprise information system to achieve high performance.

[00107] The persisted data in the persistence storage can be used as data source for enterprise data replica, provide enterprise system fail over recovery.

[00108] In what follows a detailed description about how the Transactional Audit System is incorporated into Kerberos based single-sign-on system.

[00109] **FIG 5** shows the block diagram for a Security Proxy Server **105(FIG 1)** and **FIG 6** shows the block diagram for Authentication Server **107 (FIG 1)**. The Security Proxy Server **105 (FIG 1)** in this invention is limited to processing the HTTP request only and the resources it protects are URL addressable resources.

[00110] A resource can be anything that is possible to address with a uniform resource locator (URL see RFC **1738**). A resource can include a web page, software application, services, file, database, directory, a data unit, etc. In one embodiment, a resource is anything accessible to a user on a network. The network could be the Internet, a LAN, a WAN, or any other type of network.

Table 1, below, provides examples of resources and at least a portion of their respective URL syntax:

TABLE 1

Resource	URL Encoding
Directory	/Sales/
HTML Page	/Sales/Collateral/index.html
CGI Script with no query	/cgi-bin/testscript.cgi
CGI Script with query	/cgi_bin/testscript.cgi?button=on
Application	/apps/myapp.exe
Web Services	/bytescreen/services/mywebservice

[00111] A URL includes two main components: a protocol identifier and a resource name separated from the protocol identifier by a colon and two forward slashes. The protocol identifier indicates the name of the protocol to be used to fetch the resource. Examples include HTTP, FTP, Gopher, File and News. The resource name is the complete address to the resource. The format of the resource name depends on the protocol. For HTTP, the resource name includes a host name, a file name, a port number (optional) and a reference (optional). The host name is the name of the machine on which the resource resides. The file name is the path name to the file on the machine. The port number is the number of the port to which to connect. A reference is a named anchor within a resource that usually identifies a specific location within a file. Consider the following URL:

[00112] "http://www.bytescreen.com/bytescreen/sales/index.html." The string "http" is the protocol identifier. The string "www.bytescreen.com" is the host name. The string "/bytescreen/sales/index.html" is the file name.

[00113] A complete path, or a cropped portion thereof, is called a URL prefix. In the URL above, the string "/bytescreen/sales/index.html" is a URL prefix and the string "/bytescreen" is also a URL prefix. The portion of the URL to the right of the host name and to the left of a query string (e.g. to the left of a question mark, if there is a query string) is called the absolute path. In the URL above, "/bytescreen/sales/index.html" is the absolute path. A URL can also include query data, which is typically information following a question mark. For example, in the URL:

http://www.bytescreen.com/bytescreen/sales/index.html?user=smith&dept=sales

the query data is "user=smith&dept=sales."

[00114] A service can be addressed by a URL via http://www.bytescreen.com/bytescreen/sales/index.html?service=moneyTransfor&service=accountbalance.

[00115] Although the discussion herein refers to URLs to identify a resource, other identifiers can also be used within the spirit of the present invention.

[00116] **FIG 5** shows the high level functional block of the Security Proxy Server **105 (FIG 1)**. Security Proxy Server will first process the request it received, parse it and make sure it is an HTTP request by checking each component of the request against HTTP protocol through the HTTP processor **501.** A standard web server will be able to provide HTTP processing capability. The HTTP processor is implemented within the Web Server and Web Server plug-in. The authentication processor and authorization processor can be either implemented within the Web Server plug-in or within an Application Server known in the art. If the authentication processor is built into an Application Server and The Web Server has to communicate with the Application Server via network, an SSL communication between the Web Server and Application Server has to be enforced. After the HTTP processing, the request will then be passed to the authentication processor **502** to verify if the user in the request is indeed the authentic user. If authenticated, the user's request will be passed to the authorization processor **503** to see if this user is authorized to access the resource identified by the URL. If authorized, the request will be routed to the destination web server via Router **504**.

[00117] **FIG 6** shows the high level functional block of an Authentication Server **107 (FIG1)**. The Authentication Server will first process the request it received, parse it and make sure it is an HTTP request by checking each component of the request against HTTP protocol through the HTTP processor **601.** The reason to perform the HTTP checking is that the Authentication Server is designed only to accept HTTP requests for user authentication. The standard web server will be able to provide HTTP processing capability. In the Authentication Server, the processor is

implemented within the Web Server and Web Server plug-in. The authentication processor can be either implemented within the Web Server plug-in or within an Application Server known in the art. There should be a user id and password and optionally a domain name being associated with the HTTP request for authentication. The communication over the network has to be SSL-enabled or HTTPS. If the request is valid, the user ID and password will be extracted from the request and will be passed to the authentication processor **602** to make sure the user is indeed the one as the user claims, If the user is authenticated, TGT is created that contains the user profile and will be sent back to the Security Proxy Server **105 (FIG 1)**, otherwise, an unsuccessful status is sent back to the Security Proxy Server **105 (FIG1).**

[00118] **FIG 7** shows block diagram of the authentication process. In step **701,** Client **710** sends an HTTP request to Security Proxy Server **712**. The request may need to pass firewall **720** and IDS (not shown in **FIG 7**). Initially, the request directly requests access to the URL specified. The Security Proxy Server **712** will check the request and if not authenticated, the server **712** will then forward it to the Authentication Server **713.** The Authentication Server **713** is located inside the second firewall for well-confined security, since the critical information like the user id and password for connecting to User Registry **714** will be protected if they are within the second firewall. In step **702,** the authentication request will be sent to the Authentication via HTTPS. Authentications Server **713** will communicate with User Registry **714** to authenticate the user via step **703.** The authentication status will be returned to Authentication Server by step **705,** If the user is authenticated by the User Registry, the Authentication Server will create a Kerberos token, which contains user profile and authorized roles associated with the user, otherwise, the authentication failed status will be returned. The Authentication Server **713** will return the token (only when authentication is successful) and the status to the Security Proxy Server via step **706**.

[00119] **FIG 8** depicts the Authentication Process within Security Proxy Server **105 (FIG 1).**

[00120] Step **801**: The HTTP Processor will detect what kind of HTTP method the request is. Possible HTTP Methods include but not limit to GET, POST, PUT, and DELETE. The HTTP Processor will then route the request to

the corresponding request handler and allow the request handler to do the following steps.

[00121] Step **802**: at the corresponding request handler, create a transaction ID to indicate the beginning of the transaction. A sample transaction ID is created here as: hostname:time_in_million_seconds:randomnumber, such as: Bytescreen:1073509029905:092939.

[00122] The audit-request object is defined based on the XML schema as listed in appendix 2001 and a sample XML is shown in appendix 2002. The audit-request object is also used as audit-response object.

[00123] The transaction id is added into the request object.

[00124] Step **803:** The URL is checked to see if it is a login URL. A login URL is a specific URL that is defined for sending login information. This URL is usually defined in a property file and will loaded into memory during Security Proxy Server Starting time.

[00125] Step **804**: If the request URL is a login URL, then the login is processed.

[00126] Step **812:** If the request URL is not a login URL, then the request is passed to the Authorization processor.

[00127] Step **805**: The login parameters are retrieved from the request. These parameters include: Userid, Password, Domain Name (for multi-domain single sign-on), Remote IP address (the ip address of the client), and Remote Hostname (optionally). The Userid, Domain Name, Remote IP address and Remote Hostname are logged into audit-request object.

[00128] Step **806:** A HTTP POST message is sent to Authentication Server via TCP/IP or HTTP. The HTTP POST message should be constructed in accordance with HTTP protocol, as defined in the art. In the HTTP POST message, the parameters in Step **805** need to be passed to the Authentication Server. The HTTP POST should use the URL specified in Step **803** with the followings parameters: Transaction ID, Userid, Password, Domain Name (for multi-domain single sign-on), Remote IP address (the ip address of the client), Remote Hostname (optionally) and those fields are embedded into audit-request object and passed downstream via HTTP header.

[00129] Step **807** The authentication status information is then logged into the audit-request object. The status information should contain: status code.

[00130] Step **808:** The authentication status code is checked.

[00131] Step **813**: The authentication-failed reason is logged into audit-request object.

[00132] Step **814:** The audit-request object is made persistent.

[00133] Step **815:** The client is notified of the failed code and the reason for the failure.

[00134] Step **810:** If authentication successful, the following information is extracted from the authentication result (and decrypted from the Kerberos TGT): Transaction ID, Authentication Status, Userid, Domain Name, Remote IP address, (optional) Remote Hostname, Kerberos TGT creation time and the authorized roles for the user.

[00135] Step **811**: The following information is added to the audit-request object: Kerberos TGT creation time and the authorized roles for the user

[00136] Step **816:** The Specified time-out value is obtained. This is specified during the Security Proxy Server startup time. Then the expiration time is calculated.

[00137] Step **817:** The time-out value and expiration time are logged into audit-request object.

[00138] Step **818**: The audit-request object is made persistent. In this case the XML is just written to a file.

[00139] Step **819:** If the user chooses to use a cookie based single sign-on method, an encrypted cookie is constructed containing the following: TransactionID, UserID, Domain Name, RemoteIP address, Token expired time, and User authorized roles. Those information should also be added into audit-data. Note that the option of selecting the cookie method or session method should be specified in the configuration at the Security Proxy Server startup time.

[00140] Step **820**: The encrypted cookie is sent to the client via SetCookie header, and the communication is preferably SSL-enabled for strong security but it can also be in just HTTP if the security requirement is weak. But note that the cookie can be easily hacked if just using HTTP.

[00141] Step **821**: If the user chooses not to use the cookie method, the following are saved into a session object and added into audit data:

TransactionID, UserID, Domain Name, RemoteIP address, token expiration time, User authorized roles, and SessionID.

[00142] Step **822**: The sessionID is sent to client via SetCookie header. The communication must be SSL-enabled.

[00143] As shown in **FIG 6**, the Authentication Server contains two main functional blocks: HTTP Processor and Authentication Processor. Since the HTTP Processor is generally available by standard web servers and its plug-ins, only the details of the Authentication processor **602 (FIG 6)** will be reviewed here.

[00144] **FIG 9** depicts the Authentication Processor flow chart.

[00145] Step **901:** The login parameters are obtained from the request. These parameters include: Transaction ID, Session ID, Userid, Password, Domain Name, Remote IP address, and Remote Hostname (optionally).

[00146] Step **902:** The audit-request object is obtained from HTTP header, in the example given, it is an XML String.

[00147] Step **903**: The login-count stored with the user is checked to determine if it is less than the given maximum allowed.

[00148] Step **904:** Then the user is authenticated by passing the userid and password to LDAP or Database. The communication has to be SSL or other secure communication protocol.

[00149] Step **905:** If authenticated, the roles associated with the user are obtained.

[00150] Step **906**: The Kerberos Ticket Granting Token (TGT) is created. These include: Transaction ID, Userid, Domain Name, Remote IP address, (optional) Remote Hostname, token expiration time, and Authorized roles for the user. Note that TGT is a specific type of audit data and can be set as a part of audit-request object or audit-response object for authorization uses.

[00151] Step **907**: Next the TGT is encrypted with a symmetric encryption algorithm with 128 bit or higher encryption, such as BLOWFISH.

[00152] Step **908**: The audit data are logged into the audit-response object. In the example shown, the audit-response object is the same as audit-request object, which is a XML String. One type of information that needs to be logged is the login-count.

[00153] Step **909**: The audit-response object is set into the HTTP header.

[00154] Step **910:** The result is then returned to Security Proxy Server **107(FIG 1).** The result will include: Status, Encrypted TGT, and the audit-response object embedded in the HTTP header.

[00155] **FIG 10** shows the basic requirement for the user registry. User registry **109 (FIG 1)** contains a user profile that can be stored in LDAP **110 (FIG 1)** or Database **111 (FIG 1).** Each user should as a minimum have a unique userid, a password and login-count. Additional user profile information can be added as needed. The userid uniquely identifies a user, a password authenticates the user, and the login-count records how many-failed login attempts this user has made. If the login-count exceeds the maximum allowed number, the authentication server will not allow the user to continue to try to login. This limit will protect the system from password cracking and hacking techniques. Each user is associated with zero or more roles. Each role gives the user certain privileges. For example, an Insurance company's claim adjuster can have two roles: claim adjuster and employee. Those roles are used in the authorization process to grant or deny a user's request for a particular service.

[00156] **FIG 7** also depicts a typical authorization flow. After a user is authenticated, the user will be presenting the Kerberos TGT wherever the user is trying to access the system resource **701**. One option is to put the TGT into the HTTP header and the other is to put TGT into audit data. In the present implementation of the demonstration, the Cookie header is used. When Security Proxy Server **712** receives the request, it will invoke the authorization process in **FIG 11** to authorize the user. Once the user is authorized, his request is passed to Web Server **715** together with the TGT embedded in the HTTP header downstream and in the same way to Application Server via **724**. Application Server **716** can invoke the authorization process illustrated in **FIG 11** to authorize the request and, if authorized, it will pass to MQ Server **718** via **727.** Alternatively, Application Server **716** will pass the request with the TGT embedded in the header to the Security Proxy Server **717** via **725.** The Security Proxy Server **717** will invoke the authorization process of **FIG 11** to authorize the request, if authorized, the request will be sent to the Web Service **722** via **726.**

[00157] **FIG 11** is the authorization process flowchart. At any authorization point, for example, Security Proxy Server **721** and **725**, Application Server **716** or Mainframe Server **719,** the authorization process shown in **FIG 11** can be invoked to authorize a request. The request has to carry the TGT to be authorizable.

[00158] Here are the processing steps:

[00159] Step **1101**: This is an initialization step. It will be only executed once during the authorization point startup. For example, in Security Proxy Server **721**, it will only happen once at the server startup. During this step, the encryption/decryption key for TGT and security policies will be loaded. In general, the security policy will define how the resource protected can be accessed. The polices will be cached or stored in in-memory database. In this particular implementation, the policy will contain the following information:

[00160] Resource needs to be protected. In the example, the following portion defines the protected resources:

[00161]
```
<protected-resource>
    <web-resource-collection>
    <web-resource-name>ProtectedArea</web-resource-
    name>
    <url-pattern-module>
    <url-pattern>/*</url-pattern>
    <filters>
    <inbound-process-filter-class/>
    <outbound-process-filter-class/>
    </filters>
    </url-pattern-module>
    </web-resource-collection>
    </protected-resource>.
```

[00162] The roles that are allowed (or denied) for access to this resource.

[00163] Rules can also be defined as part of the policy. For example, rule1: only users have role1 are granted Rule2: without role 2, cannot be granted access. In general, rules will combine user profile, policies, and business related information to help make resource accessibity decision.

[00164] In the XML example, the portion defines the roles that will be used in the policy rule:

```
<auth-constraint>
<role-name>adjuster</role-name>
<role-name>employee</role-name>
</auth-constraint>
```

[00165] The above sections of XML simply indicated that users with role "adjuster" or "employee" can access the resource /* (everything under the root /)

[00166] The policy needs to be loaded from the secured area: namely, Authentication Server **107 (FIG 1)**

[00167] Step **1102**: The TGT is retrieved. This token is passed downstream by HTTP header: Cookie. Only the token from Cookie header is passed. If the TGT is passed down by other means, for example, in MQ Server, the token may be passed by JMS Header as filed or even in the message body, and just retrieved as specified. Since the TGT is encrypted, it must be decrypted when it is obtained. Then the following is extracted from the token: Transaction ID, Userid, Domain Name, Remote IP address, (optional) Remote Hostname, token expiration time, and Authorized roles for the user.

[00168] Step **1103**: Authorized roles for the user are checked against the roles and rules in the policies that carried in either audit data or in-memory cache, which are based on the policy rule. If not authorized, "authorization-failed" status is returned. If authorized further, the expiration time is checked against current time. If time test is satisfactory, "authorization successful" status is returned.

[00169] Step **1104**: The audit-request object is obtained either from HTTP header or message body, and the following information logged into audit-request object: Service-id (can be resource such as a URL, or any given service name), Authorization status (SUCCESS or FAILED code), failed reason (if failed), policies applicable to the resource, roles used for access, and additional business related log info (if any).

[00170] Step **1105** The authorization status is checked. If "yes," then the system proceeds to **1105;** if "no", then go to **1108.**

[00171] Step **1106:** The audit-request object is made persistent and sent to the persistence storage device if necessary.

[00172] Step **1107**: Access is allowed to the resource and the request is passed downstream.

[00173] Step **1108:** An audit-response object based on audit-request object is created. In the implementation here, the audit-request object – XML string – is duplicated.

[00174] Step **1109**: The audit-response object is made persistent and forwarded to persistence storage if necessary.

[00175] Step **1110:** The audit-response object is added to the response.

[00176] Step **1111**: The response is then returned to the parent process system.

[00177] **FIG 12** depicts a customizable plug-in module **1204** and **1205** in Security Proxy Server **1201**. This plug-in allows dynamic loading of an InBoundFilter **1204** and an OutBoundFilter **1205**. The InboundFilter will allow the Security Proxy Server to process the HTTP request before being sent out the destination resources and the OutBoundFilter will process the HTTP response before it is sent back to the client.

[00178] These customizable plug-ins allow the Security Proxy Server to be customizable to satisfy specific needs. For example, it can be used to perform data validation, data integrity checking, XML transformation and provide user profile, security prolicies and buiness information data binding. One important usage is to plug user abstraction service or user profile retrieval and Kerberos based authentication/authorization via vender provided API abstraction or service to delegate authentication/authorization process into proxy server to provide Identity Management vendor independent infrastructure for the enterprise. With this layer of user and API abstraction, the security infrastructure will be able to work with all Identity Management Systems vendors without changing its security infrastructure. The following contains a description of a method to pass the audit-request object and audit-response object described in the Transactional Audit System to the SOAP request message and SOAP response message for Web Service technology.

[00179] Here are the steps to accomplish this:

[00180] Step 1: The audit-request object and audit-response object are defined with XML schema as shown in the example in appendix 2001. Lets refer it as audit schema. The example in appendix 2001 is only one example. The schema can be defined to meet each business' needs.

[00181] Step 2: The audit schema in included into the schema definitions for the Web Service input request message and output response message. The root element in the audit schema should be included as one element in the input request message schema and also in the output response message schema.

[00182] Step 3: The client code and server code are then generated as specified in web service technology, as known in the art, with the updated input request message schema and output the response message schema.

[00183] Step 4: The client **1200** code is then modified in order to pass the userid, password via either GET or POST request parameters or via HTTP headers to precede authentication against Security Proxy Server **1201.**

[00184] Step 5: Once authenticated as described in the Authentication Process Earlier, the Security Proxy Server **1201** will send the TGT to the Client **1200** via HTTP header such as SetCookie.

[00185] Step 6: The Client will use HTTP header to pass the TGT for subsequent access request.

[00186] Step 7: The Security Proxy Server **1201** will invoke the authorization process to authorize the request.

[00187] Step 8: Once authorized, the Security Proxy Server **1201** will begin to send the request to the destination web service

[00188] Step 9: Before the request leaves the Security Proxy Server **1201**, the InBoundFilter **1204** will intercept the request (SOAP Message) and update the audit-request object that was sent by the Client **1200** with the audit-request object returned from the authorization process into the SOAP Message in the request.

[00189] Step 10: At the Web Service **1202,** the web service implementation will be able to get the audit-request object from the request message and be able to add the logging information to it.

[00190] Step 11: Web Service Implementation should transfer the logging information from the audit-request object in the request to the audit-response object in the response.

[00191] Step 12: When the response comes back, the OutBoundFilter will intercept the response and extract the audit-response object from the SOAP message and update it (or optionally *replace* it with a nil-valued audit-response object).

[00192] Step 13: The extracted audit-response object will be passed to the client via HTTP header or by other processing devices such as another application server, as specified by the client.

[00193] Another scenario important to point out is that the method presented here in securing Web Services does not treat Web Service any differently than any other web resources. That means, unlike the products currently available, the invention is a uniform method in dealing with all web resources, including web service. That means that the audit information regarding activities of web services will be logged by user, not by the identity of the service that the user uses to access the web service. This point was also demonstrated in the descriptions of **413** and **414** in **FIG 4**.

[00194] In the foregoing descriptions, a Transactional Audit System has been described that provides centralized security audit for enterprise systems. The Transactional Audit System is then incorporated into the Kerberos based multi-domain single sign-on system to provide authentication, authorization, access control and centralized audit. The covered areas range from firewall, IDS, security proxy server, web server, application server, web services, MQ server, Mainframe and external network access.

APPENDIX:

2001 The sample xml schema that defines the audit data structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Jun Song -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="AAA-ARoot">
        <xs:annotation>
            <xs:documentation>Comment describing your root element</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="transaction-id" type="xs:string"/>
                <xs:element ref="user-profile"/>
                <xs:element ref="authentication-audit"/>
                <xs:element name="life-span">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="creation-time" type="xs:dateTime"/>
                            <xs:element name="expiration-time" type="xs:dateTime"/>
                            <xs:element name="timout" type="xs:int"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element ref="audit-trail"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="user-profile">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="userid" type="xs:string"/>
                <xs:element name="user-registry-domain" type="xs:string"/>
                <xs:element name="remote-connection-info">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="hostname" type="xs:string" minOccurs="0"/>
                            <xs:element name="ip-address" type="xs:string"/>
                            <xs:element name="additional-info" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element ref="roles"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="audit-trail">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="service-log" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="service-id" type="xs:string"/>
                            <xs:element name="authorization-status" type="xs:string"/>
                            <xs:element name="faild-reason" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                            <xs:element name="accessing-role">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element ref="role" maxOccurs="unbounded"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="in-coming-request">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element ref="name" minOccurs="0"/>
                                        <xs:element ref="log-info"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="out-going-request" minOccurs="0"
maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element ref="name"/>
                                        <xs:element ref="log-info"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                    <xs:element name="in-coming-response" minOccurs="0"
maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="name" minOccurs="0"/>
                                <xs:element ref="log-info"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="out-going-response">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="name" minOccurs="0"/>
                                <xs:element ref="log-info"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="additional-checkpoint" minOccurs="0"
maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="name" minOccurs="0"/>
                                <xs:element ref="log-info"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="roles">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="description" type="xs:string" minOccurs="0"/>
            <xs:element ref="role" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="datetime-format" type="xs:dateTime"/>
```

```
<xs:element name="string-format" type="xs:string"/>
<xs:element name="log-info">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="time-stamp" type="xs:dateTime"/>
            <xs:element name="message" type="xs:string" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="name" type="xs:string"/>
<xs:element name="role">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="description" minOccurs="0"/>
            <xs:element ref="role-name"/>
            <xs:element ref="role-reference" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="description" type="xs:string"/>
<xs:element name="role-name" type="xs:string"/>
<xs:element name="role-reference">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="reference" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="authentication-audit">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="authenticsation-status" type="xs:string"/>
            <xs:element name="faild-reason" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

2002 Sample Audit-request object based on the schema listed in 2001.

```
<?xml version="1.0" encoding="UTF-8"?>
<AAA-ARoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\WebSecure\jssw\config\core\aaaaroot..xsd">
```

```
<transaction-id>www.bytescreen.com:30405050606:02020304</transaction-id>
<user-profile>
    <userid>jsong</userid>
    <user-registry-domain>bytescreen</user-registry-domain>
    <remote-connection-info>
        <ip-address>203.245.23.124</ip-address>
    </remote-connection-info>
    <roles>
        <description>Employee</description>
        <role>
            <role-name>employee</role-name>
        </role>
    </roles>
</user-profile>
<authentication-audit>
    <authenticsation-status>success</authenticsation-status>
    <faild-reason></faild-reason>
</authentication-audit>
<life-span>
    <creation-time>2004-01-01T00:00:00</creation-time>
    <expiration-time>2004-01-01T00:20:00</expiration-time>
    <timout>120</timout>
</life-span>
<audit-trail>
    <service-log>
        <service-id>claimservice</service-id>
        <authorization-status>sucess</authorization-status>
        <accessing-role>
            <role>
                <role-name>employee</role-name>
            </role>
        </accessing-role>
        <in-coming-request>
            <log-info>
                <time-stamp>2004-01-01T00:00:02</time-stamp>
                <message>enterred access-patient-infomation</message>
            </log-info>
        </in-coming-request>
        <out-going-response>
            <log-info>
                <time-stamp>2004-01-01T00:00:05</time-stamp>
                <message>exit access-patient-infomation</message>
            </log-info>
        </out-going-response>
    </service-log>
</audit-trail>
</AAA-ARoot>
```

2003 Sample Policy XML

```
    <security-constraint>
        <display-name>Example Security Constraint</display-name>
        <protected-resource>
            <web-resource-collection>
                <web-resource-name>Protected Area</web-resource-name>
                <url-pattern-module>
                    <url-pattern>/*</url-pattern>
                    <filters>
                        <inbound-process-filter-class/>
                        <outbound-process-filter-class/>
                    </filters>
                </url-pattern-module>
            </web-resource-collection>
        </protected-resource>
        <excluded-resource>
            <web-resource-collection>
                <web-resource-name>Exclueded Area</web-resource-name>
                <url-pattern-module>
                    <url-pattern>*.jar</url-pattern>
                </url-pattern-module>
                <url-pattern-module>
                    <url-pattern>*.jpg</url-pattern>
                </url-pattern-module>
                <url-pattern-module>
                    <url-pattern>*.gif</url-pattern>
                </url-pattern-module>
                <url-pattern-module>
                    <url-pattern>*.js</url-pattern>
                </url-pattern-module>
                <url-pattern-module>
                    <url-pattern>*.css</url-pattern>
                </url-pattern-module>
                <url-pattern-module>
                    <url-pattern>*.txt</url-pattern>
                </url-pattern-module>
                <url-pattern-module>
                    <url-pattern>*.swf</url-pattern>
                </url-pattern-module>
                <url-pattern-module>
                    <url-pattern>*.mspx</url-pattern>
                </url-pattern-module>
                <url-pattern-module>
                    <url-pattern>*.pdf</url-pattern>
                </url-pattern-module>
            </web-resource-collection>
        </excluded-resource>
        <auth-constraint>
            <role-name>adjuster</role-name>
            <role-name>employee</role-name>
        </auth-constraint>
    </security-constraint>
```